Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 508 – 517

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

Effectiveness of Hybrid Workflow Systems for Computational Science

Beth Plale*, Eran Chinthaka Withana*, Chathura Herath*, Kavitha Chandrasekar*, Yuan Luo*

School of Informatics and Computing
Indiana University, Bloomington

Abstract

The workflow and its supporting systems are integral to computational science. Tailored to loosely coupled, and largely coarse-grained tasks, the workflow replaces the script as a way to automate the multiple steps of a large scale model. Workflow reuse has been at the subworkflow level but this restricts, over the long run, a workflow to running on the system on which it was developed. A scientist wanting to use two workflows developed by two different people and for different workflow systems will need to have access to both workflow systems. The contribution this paper makes is a qualitative and quantitative study of the tradeoffs of a hybrid workflow solution that utilizes multiple workflow systems and solutions to execute a single workflow. Our results indicate that the major tradeoffs are not in performance as much as they are in complexity.

1. Introduction

The workflow and its supporting systems are integral to computational science [1]. Tailored to loosely coupled, and largely coarse-grained tasks, the workflow replaces the script as a way to automate the multiple steps of a large scale model, including moving data, identifying and locating input data sources, transforming data from one form to another suitable for model digestion, and assimilation of data from multiple sources into a single form.

Workflow systems often provide default activities that serve as out-of-the-box tasks that can be used to construct a workflow. These activities may be domain independent, such as third party data movement, or targeted towards a particular science domain such an activity that carries out a BLAST gene sequence matching activity [2]. The choice of workflow activities supported by a system often depends on the targeted use of the system. A workflow system might be specialized to work extremely well when tasks are jobs that run on large-scale compute resources such as Teragrid/xSEDE [3], or a cloud platform. Sharing of workflow activities through sites such as myExperiment.org [4] further enhance the usefulness of a particular workflow system by allowing domain specific sets of tasks to be constructed by one into a workflow or subworkflow and then shared with many. But workflow scripts usually run on a single workflow engine.

*Corresponding author

Email addresses: plale@cs.indiana.edu (Beth Plale), echintha@cs.indiana.edu (Eran Chinthaka Withana), cherath@cs.indiana.edu (Chathura Herath), kavchand@cs.indiana.edu (Kavitha Chandrasekar), yuanluo@cs.indiana.edu (Yuan Luo)

doi:10.1016/j.procs.2012.04.054

Workflow reuse has been at the subworkflow level as has been noted by the stewards of the myExperiment workflow repository. That is, users share portions of workflows, and these workflows are being picked up for adoption at higher rates than are full workflows. For simplicity in exposition, we henceforth refer to a sub-workflow simply as a 'workflow'. Since workflow scripts usually run on a single engine, it is likely that any long-term guarantee that a workflow system makes to reproducibility of its workflows will be specific to the system on which the workflow was first created. But this limits over the long run a workflow to running on the system on which it was developed. A scientist wanting to use two workflows developed by two different people and for different workflow systems will need to have access to both workflow systems.

Hybrid computing of the form of utilizing more than one workflow system to carry out a single task has been explored in the WS-VLAM system [5]. WS-VLAM integrates support for multiple workflow systems through a single event service bus, the VL-e Workflow Bus, that carries data and control to and from multiple remote workflow systems. Such an approach seems highly desirable, but what is the cost? We explore that question fully in this paper.

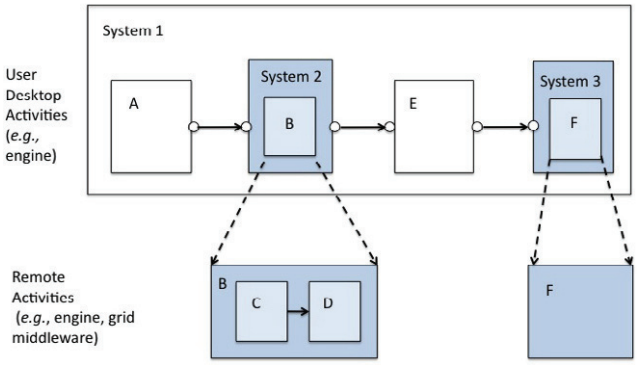


Figure 1: Dimensions of hybrid workflow systems. Activities *B* and *F* are called from System 1 by instructing Systems 2 and 3 respectively to execute the activities. *B*, while seen as a single activity in System 1, is actually a workflow. *F* is an activity that runs on grid middleware. Diagram adapted from [6].

We undertake a performance evaluation of hybrid computing of the kind demonstrated by WS-VLAM that involves multiple workflow systems and evaluate the approach qualitatively and quantitatively. To our knowledge there is no good comparative data on the costs, both quantitative and qualitative, of what it takes to support such a strategy. Our study fixes the high level system at a user desktop workflow engine, and explores the performance and programming impact of various forms of remote activity. The structure adopted is best illustrated in Figure 1. System 1 is hosted on a user's local machine, and workflows are run locally. Activities *A* and *E* run locally. Activity *B* is a call from System 1 to System 2 to execute the activity. *B*, while seen as a single activity in System 1, is actually a workflow made up of activities *C* and *D*. Activity *F* is executed by a call from System 1 to System 3, instructing the latter to execute the activity. *F*, which is seen as an activity in System 1 calls out to grid middleware to carry out the execution of the activity.

In our evaluation we hold System 1 constant at the local desktop workflow system, specifically, the Trident Scientific Workflow Workbench [7] running on top of Workflow Foundation. Trident was chosen because it is easy to write new activities, and because as a Windows solution, it brings with it platform diversity, since System 2 and System 3 are mostly Linux in our setup. In more detail, using the model in Figure 1 to illustrate, for the System 2 case we evaluate both the Kepler workflow system [8], and the Apache ODE [9] workflow tool. Trident is well suited to a desktop environment, hence was chosen for System 1. Kepler is often used to run workflows on the grid and Apache ODE is widely used in enterprise environments. At System 2, these two workflow systems represent two categories of workflow systems [10], as Kepler uses Graph-based modelling and ODE with it BPEL-based orchestration follows Language-based modelling for workflow composition. With regards to Workflow scheduling, all the three workflow systems use a centralized scheduling scheme. This taxonomy allows our results to be transferrable to other workflow systems. For the System 3 case where remote services are invoked directly we evaluate GFac [11] and the Opal toolkit [12].While GFac dynamically provides a generic interface to services on the grid, Opal wraps an application statically to expose an interface specific to the application. These two examples cover the typical wrapping methods

for applications on grid.

The contribution this paper makes is a qualitative and quantitative study of the tradeoffs of a hybrid workflow solution. As our results indicate, the remote engine model compares favorably to the remote grid middleware in terms of performance. The complexity of maintaining such a hybrid system may in the end outweigh any benefits that could be accrued from such a flexible solution.

The remainder of this study is organized as follows. Section 2 discusses related work. Section 3 details architectural aspects. Section 4 discusses the workload and Section 5 discusses the quantitative and section 6 qualitative results. Section 7 discusses future work.

2. Related Work

A wide range of workflow systems [10, 13] have been developed for scientific research and their maturity and established user base influenced the design constraints of this study. While most systems are general purpose [10], others have a stronger domain focus [2]. Triana [14] targets complex workflows for domains like signal, text and image processing. Some of these systems are optimized for supercomputing resources (e.g. Grids [10], Dagman [15], Wings [16]). Pegasus [17], in combination with DAGMan [18] treats parametric sweeps as first class operations. Others are better suited to desktop or single user environments such as Taverna [2] and Trident [7].

Researchers have examined ways to standardize and evaluate workflow systems. Jinde et al. [19] study interoperability amongst business workflows management systems. There have been efforts to standardize workflow interoperability; Hayes et al. 2002 [20] proposes standards to which the workflow systems adhere to achieve business workflow interaction, yet these standards imposed limitations that arise in a particular business setting. The AFRICA [21] framework supports interoperability among workflow systems using SOA based XML message interactions. Though these efforts are promising, different workflow representations (e.g. directed graphs, petri-nets, UML diagrams), unique domain and compute resource specific requirements within different workflow environments have complicated attempts at interoperability. Select workflows systems [2], [22] coordinate workflow tasks designed for another workflow system.

Performance evaluation often examines an aspect of performance, such as Wang et al. [23] performance evaluation of virtual machine-based Grid workflow system. Stratan et al. [24] evaluate grid workflow engines on characteristics like overhead, raw performance, stability, scalability and reliability on different workloads. We compare two models based on overall performance and overhead quantitatively. Gillmann et al. [25] benchmark workflow systems based on e-commerce scenarios found in enterprises. Truong et al. [26] define the various granularities at which a grid workflow system can be evaluated. Our study is focussed on the strengths of layered workflow engines and services.

3. Architectural Organization

The architectural organization is of the infrastructure used in the study, not of a working infrastructure. It can be viewed as a combination of cases, a few of which might be used in practice. While exercising the whole architecture may be unrealistic in a real setting, the cases individually and in small-group combinations are realistic. There are four primary components of the architecture, which are refinements to the model of Figure 1 and these are:

Baseline execution: The baseline execution does not employ hybrid workflows, and instead runs workflows locally, within System 1 in Figure 2. The top level workflow engine is the Trident Scientific Workflow Workbench and it runs workflows locally. Trident calls out to the Workflow Foundation to orchestrate execution.

Remote workflow engine: where an activity in the local system contacts a remote workflow engine, illustrated by activities A_R and A_T in the figure.

Remote grid/cloud middleware: where activities in the local system contact grid/cloud middleware directly. Since there is no orchestration at the remote system, the remote grid/cloud services are capable of executing only one task of a workflow. Shown in the black dotted lines is GRAM executing one of the three services of a workflow.

High performance compute resources are the high performance compute resources such as Teragrid, OpenScienceGrid, or a supercomputer.

To illustrate the distinctions, using terminology of Figure 1, baseline execution is carried out by Trident running on a Windows machine and using Workflow Foundation. Trident activities A_p and A_q invoke two workflow tasks under the control of Workflow Foundation running on the same host or host cluster as Trident.

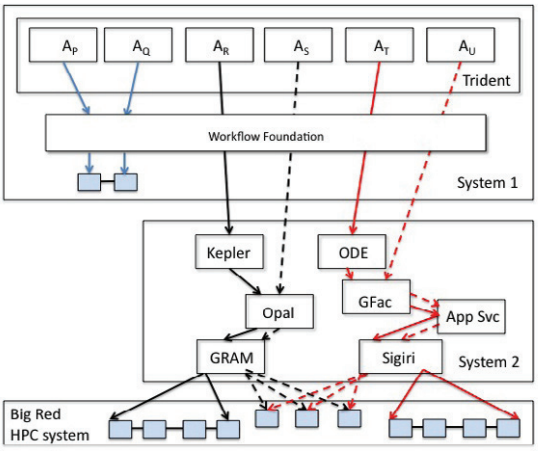


Figure 2: Types of interoperability: local execution within System 1 is shown by activity A_P and A_Q invoking a workflow through WorkflowFoundation. Activity A_R communicates with the Kepler remote engine to run a subworkflow on Big Red (solid black lines). Activity A_S contacts grid services directly to invoke nodes individually (dotted black lines). Activity A_T invokes the ODE workflow engine to run a subworkflow on Big Red (solid red lines). Activity A_U contacts grid services directly to invoke nodes individually (dotted red lines).

The remote workflow engine case is illustrated by activity A_R's invocation of the Kepler workflow system to orchestrate a workflow. Kepler uses the resource manager, Opal, to submit jobs to the supercomputer, Big Red; Opal submits using Globus GRAM [27]. The remote grid/cloud middleware case is A_S initiating a remote grid/cloud service directly. Cases 4 and 5, A_T and A_U, are second examples of cases A_R and A_S respectively. A_T invokes the Apache ODE workflow engine which schedules jobs using GFAC, and WS-GRAM [27] with job communication facilitated through a wrapper service that allows a web services infrastructure to communicate with legacy Fortran codes. A_U invokes remote grid/cloud services directly.

A_R invokes Kepler which contacts the Opal Toolkit to execute a subworkflow on the Big Red supercomputer through Globus Gram. Activity A_T invokes the Apache ODE workflow engine that contacts GFac to execute a subworkflow on Big Red using Sigiri resource manager [28].

4. Workflow workloads

The core workflows of the study cover four cases over two workflow patterns as follows:

1. Data Intensive, Sequential Workflow
2. Data Intensive, Parallel Workflow
3. Compute Intensive, Sequential Workflow
4. Compute Intensive, Parallel Workflow

The sequential workflow is made up of 5 identical tasks executed sequentially. For the compute-heavy version Linpack Java version is used. Linpack solves linear equations, and we configured it to carry out execution on a 5000 x 5000 matrix, using double precision floating point coefficients. The inputs and outputs are small, on the order of 5KB. The data-intensive workflow carries out applies a cryptographic hash function (i.e., MD5) on a 1.5 GB file. Because there are no natural data dependencies in this workflow, a 1.5 GB file is copied from one location to another before each MD5 service is performed. The parallel workflow executes the same computations (MD5 or Linpack) in parallel. The workflow is sandwiched on either end by scatter and gather nodes.

A sequence diagram shown in Figure 3 temporally depicts activity along a vertical timeline of invocation sequences for the Kepler/Opal stack, showing communication between workflow engines and grid services.

5. Evaluation

The experimental performance evaluation is focused on exposing performance overheads of the hybrid model, and does so through the following metrics:

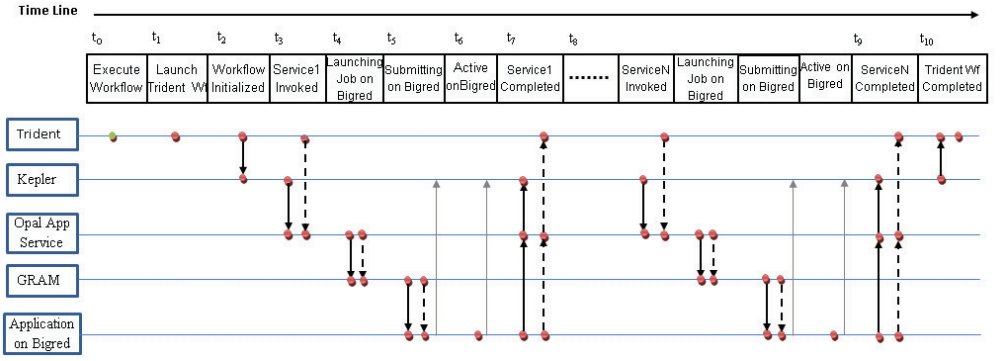


Figure 3: Sequence diagram showing workflow engine interaction between Trident and Kepler. Solid lines are events, dotted lines are remote grid services events, and gray lines are notifications.

1. Overhead of remote engine execution. Measured by the time to invoke workflow instance on local machine versus time to invoke workflow through a remote workflow engine. This is the cost in time incurred by using a second workflow engine.
2. Difference between workflow execution through a second workflow engine versus directly accessing remote grid services.
3. Variability in approaches for subworkflow stacks: gives variation for given service time or latency. Best and worst case can often vary significantly.

The test environment has the Trident Workflow Workbench and the Workflow Foundation running on a Windows 2008 R2 Enterprise edition server. The underlying machine has an Intel E7540 CPU, which consists of four 2.0 GHz processors, each with six cores (24 cores total); the machine has 128 GB RAM. We use Trident version 1.2.1 and Workflow Foundation 4. The remote services, shown as System 2 in Figure 1, are run on a cluster consisting of 16 dual-socket, 2 core (4 total cores/node) MD Opteron system with 16 GB of memory per node running 64-bit Red Hat Enterprise Linux. Nodes are connected via Gigabit Ethernet. Local disks are 73 GB and nodes in the cluster have GB Ethernet connectivity to a 24 TB NAS that is front-ended by a file server running NFS v4.

The remote workflow tasks are executed on a 1024 IBM JS21 Blade server (IU Big Red), each with two dual-core PowerPC 970 MP processors, 8GB of memory per node at Indiana University. Compute nodes are connected via Gigabit Ethernet to a GPFS file system hosted on 16 IBM p505 Power5 systems. We submit jobs to run on the system through IBM's LoadLeveler resource manager. LoadLeveler, in turn, relies on Adaptive Computing's Moab scheduler to dispatch user jobs to appropriate and available compute nodes. In our experiment, we use BigRed queue set up for experimental use. At 4 node, 16 core/node capacity, it is small but offers low queue latencies.

The data intensive workflows make use of the underlying Lustre Distributed File System for file movement within the same site. The workflow suites used in the evaluation are capable of moving files between remote sites using GridFTP and RFT, but the data intensive workflows in this evaluation focus on data movement within the same site. Every activity in the Data Intensive Sequential Workflow copies the data file to its input working directory before it starts processing and once it finishes the processing copies the output file to its output working directory. So every activity in Data Intensive Sequential Workflow will copy its input file from the output working directory of the previous activity. The first activity would copy the input file from the input parameter to the workflow.

The Data Intensive Parallel Workflow is launched with the location of the data files as the inputs to the workflow. Each activity in the workflow copies the input data files to its input working directory and produces its output files to its output working directory. The last activity of the Data Intensive Parallel Workflow gathers all the outputs from the parallel activities to a single output directory. Each test is executed five times and minimum, maximum and median values are computed.

The secondary workflow systems are Apache ODE workflow engine and the Kepler workflow system. XBaya [29] mediates between Trident and the Apache ODE workflow engine. ODE uses GFac to manage interactions with application services. GFac instantiates workflow tasks if needed; it moves data for Grid resources and interacts with

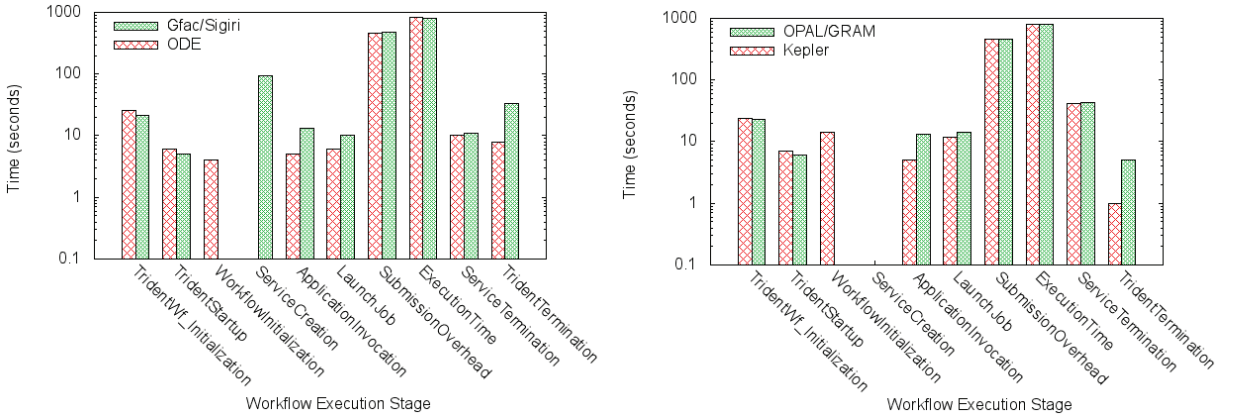
job submission and resource management providers to schedule jobs to Grid resources. We set up our experiment so that GFac uses Sigiri to submit jobs to the HPC resource. Kepler uses Opal actors to interact with job submission components. We configure Opal to use GRAM actors to submit and manage jobs. Kepler invokes an Opal actor which is a generic web service client to launch job and query job status. The Opal actor submits a job to the Opal server which submits the job to the resource manager in BigRed. Once a job is done, the output data is staged from BigRed to Opal Server. When the Opal job status is COMPLETE, the actor will be terminated.

The remote grid services we use are GFac and Opal. Trident interacts directly either with GFac or Opal server to get the jobs scheduled and executed. Trident also uses a custom file movement utility, implemented as an Opal service, for the file movement to the supercomputer resources, in the Opal-based workflows case. In the case of GFac/Sigiri remote grid services, GFac serves as data mover.

5.1. Baseline execution case

We capture the baseline measurement of workflow overhead by running a workflow under Trident and on Trident's host machine, then compare that against the same workflow executed by a secondary workflow engine. In order to have a fairer comparison, we ignore waiting time in the job queue in the remote case.

For the compute-intensive workflows, the remote ODE workflow engine instance saw only 2.4% higher execution time than local execution. The remote grid service instance using GFac/Sigiri, on the other hand, had 15.5% higher execution time than local machine execution. This difference is because of the overhead in invoking GFac, the service factory, for dynamic web service creation. Kepler workflow and Opal/GRAM grid services showed only 5% higher execution time than local machine execution. The difference in time can be attributed to the difference in architecture between local and workflow and remote grid execution, namely, the overheads for invocation of remote workflow engine, application scheduling and invocation on the grid. The 10.5% higher overhead for GFac/Sigiri versus Opal/GRAM can loosely be seen as the overhead of dynamic service creation. The service creation has to be done manually for Opal before any service invocation. Because of this the overhead of Opal invocations does not include service creation. In order to ensure network latencies did not influence our measurements, logs for remote invocation were constantly monitored for anomalies. To ensure availability of the remote engine during the experiments, it was set up as a persistence service, and monitored for availability. Also, the queue wait times were measured and removed from the overhead to remove the effect of varying queue wait times. Other side-effects like degree of utilization were prevented by setting up the remote engines as a dedicated service for experimentation.



(a) ODE and GFac/Sigiri: compute intensive workflow

(b) Kepler and Opal/GRAM: compute intensive workflow

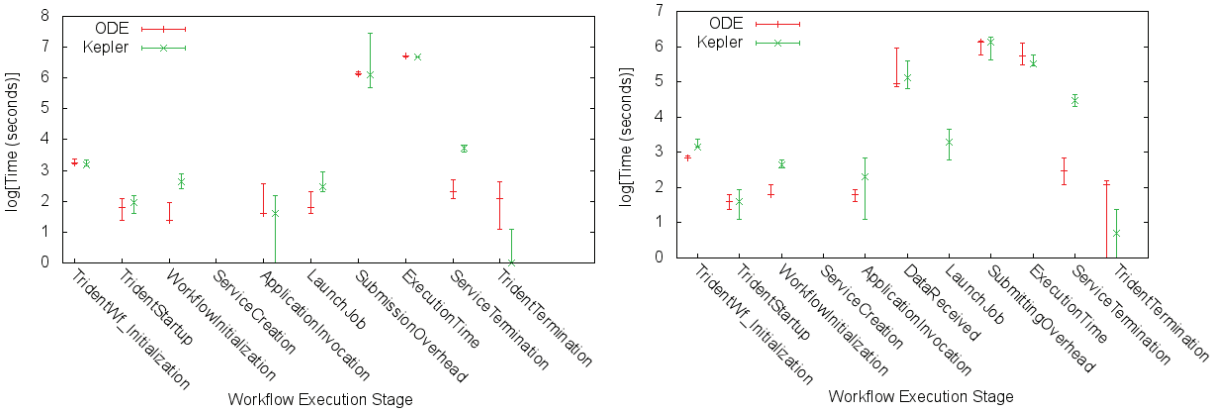
Figure 4: Compute intensive workflows

5.2. Remote access case

Recall that we have four remote cases two where Trident works with a secondary workflow engine (Kepler and ODE), and two where Trident executes a remote grid task directly using Opal or Sigiri. Service time for all four cases

broken out by major workflow stages is shown Figure 4. In Figure 4(a), the Gfac/Sigiri case has notable Service Creation and Trident Termination latencies. The ODE stack has the flexibility to reuse a dynamic service, but without ODE, Trident must ask GFac to dynamically start a new service for every task invocation. Too, in the Gfac/Sigiri case, Trident has more activities to manage. This higher number of workflow activities within Trident contributes to the latency captured as Trident Termination. Since ODE uses an internal data movement tool (*i.e.*, GFac) and every component in the synthetic workflow consumes the same input file, the tools move the file for the first activity and re-use the same location for the other nodes in the workflow. But for the Gfac/Sigiri case, files are moved for every service invocation adding more overhead to the workflow execution.

In Figure 4(b), performance of Kepler is compared to Opal/GRAM remote grid task case for the compute intensive workflow. Unlike GFac which create services dynamically, Opal uses pre-deployed web services and hence Service Creation overhead exists neither in Kepler-Opal workflow nor in the Opal/GRAM case. In the Opal/GRAM case, Trident has more activities to manage and track. It can be observed that the total execution times shown across the four case in Figure 4 all lie within a range of 800-820 seconds for all four invocation types. The data intensive workflow can be found in a longer technical report.



(a) ODE and Kepler: compute intensive workflow, sequential

(b) ODE and Kepler: data intensive workflow, sequential

Figure 5: Execution time variability per execution step.

5.3. Execution variability

An important aspect of system performance is the variability in performance observed over repeated runs. Figure 5 captures variation in execution times for ODE and Kepler solutions under both the sequential compute and sequential data workflows. The error bars represent the logarithm of minimum and maximum values of a given measurement and the plotted value represent the logarithmic median of the measured values.

The workflows on the ODE and Kepler stacks take approximately the same amount of time (16 sec), but examining the deviation between minimum and maximum values, the ODE stack has a smaller deviation (106 sec) to Kepler's (1481 sec). The large deviation of the latter can be attributed to submission overheads. The ODE stack keeps the submission overhead within a small range. Kepler uses GRAM as its job submission middleware and GRAM takes a varying amount of time to get the job scheduled and executed in the compute resource. During this experiment we also experienced job failures caused by different GRAM failures and had to re-run our experiments on multiple occasions. Additionally, the "Application Invocation" overhead in ODE is significantly lower than other stacks. Higher application invocation overhead in Kepler could be attributed to inefficient workflow activity and service handling within the Opal actors. For data intensive sequential workflows the ODE and Kepler stacks show similar performance numbers (80 sec). Kepler shows the lowest deviation (575 sec) with ODE at 650 sec. Opal stack shows significant overhead during the shutting down phase of the service (Service Termination). Further investigation revealed that this is due to the implicit log file movements in Opal server.

In addition to the overhead, the overall performance in a grid environment is influenced by the degree of utilization of grid. In case of submission to a grid, we had a reservation on a queue and the queue wait times showed a standard deviation of only around 8 seconds, for an average queue wait time for 460 seconds. As for the execution time, we observed a very low standard deviation of only 7 seconds for an average execution time of 810 seconds.

6. Qualitative Aspects of Hybrid Model

Hybrid models of interoperability can also be interestingly examined on qualitative aspects, and do that in this study examining several dimensions: 1.) Model of computation - execution model supported by a system, 2.) Level of control at desktop, and finally 3.) Complexity of the system. Each is discussed in sections below.

6.1. Model of Computation

The Model of Computation (MOC) [6, 30] captures the interaction between activities. Intuitively, the MOC gives interpretation to the edges that connect two vertices of a workflow graph or between workflow systems. Elmroth et al. [6] state that "sub-workflows are seen as black boxes and their internal MoC is not important to workflows higher up in the hierarchy", meaning that we need not consider the internal edges of the subgraph (sub-workflow). But the black-box nature of the workflow model has advantages and disadvantages. The advantage can be seen in Trident. Trident is a control-flow workflow system. All scheduling decisions are based on static information and this information is used to generate an Actor/activity firing schedule in the form of a Windows Workflow Foundation run-time script before it starts execution. But the disadvantage of a black box model is lack of control and problem solving strategies over the entire workflow if something goes wrong. Too, how integrated is the sub workflow into the top level workflow script? Are the semantics of the input and output edges of the sub workflow representable at the higher level. We did not attempt to quantify the issues with semantics and error propagation during the course of the evaluation, but clearly recognize how important the problem is to simplifying the overall user experience.

Finally, for a more extensible system, the lower level workflow systems should use a uniform interface when communicating with the top level workflow system. In WS-VLAM [5] this is accomplished by means of a common event bus, VL-e workflow bus. We did not implement this in our testbed.

6.2. Level of Control at Top Level System

What are the limits to a layered workflow system model? There are several we note here. Workflow engines often depend on select services to carry out tasks needed during workflow execution, and can be limited by this dependency. These tasks include data movement services, authentication and authorization frameworks, job submission and monitoring services. This functionality can limit the generality of the workflow engine, because it can only perform the kinds of data movement it supports. But through the remote grid model, a user can use any data movement framework she chooses and also has the ability to optimize data movements and data placement in compute resource.

If the top level workflow engine supports checkpointing and recovery operations, when a workflow fails at a certain node, the workflow can be restarted from that node. But if the failed node represents a multi-task workflow, the failure can take time to recover. For example, if a certain experiment contains 10 tasks to be executed and if it is implemented as set of components, then a failure at the 8th node is not excessively costly, because the workflow can be restarted to run at the 8th node. But if the same experiment is implemented as a workflow, and if the 3rd to 8th component are in the subworkflow, failure of the 8th node requires the workflow to be restarted from the 3rd task.

The ODE workflow engine, particularly through its instantiation in the OGCE tool suite [31] is limited by its ability to add new workflow activities. For functionality to be added, it must be exposed as a Web service. Both Kepler and Trident workflow engines on the other hand are primarily targeted to desktop based workflow executions and thus provide flexibility to incorporate new functionality easily into the system. With the actor model in Kepler and activity model in Trident, a user can program any functionality into the workflow, enabling it to support a wide variety of functionality.

In experience in the research lab and in the classroom, Trident is easy to use. Programming new workflow activities requires writing small C# activities. With its user-friendly interface and programming model, we believe the tool could appeal to the scientist who is comfortable with a Windows platform and inclined to write and deploy new workflow functionalities that execute within their local compute resources. We conducted a heuristic evaluation [32] of Trident,

Swift, VisTrails, Kepler, Taverna, and Ode and found that Trident had the lowest time to get a custom workflow up and running.

6.3. Architectural Simplicity

The remote grid service approach, in contrast to the remote workflow engine approach, gives the user more control over the execution of the experiment. But with this control comes maintenance overhead and complexity, because the user has to manage all the components and their interactions. For example, if any of the interfaces to the remote tasks change, the workflow author has to update the components to account for these changes. In the remote workflow engine solution, these changes would be handled inside the calling workflow engine infrastructure. From our experience with the LEAD [33] science gateway, grid middleware adds complexity to the architecture. Handling security issues is also a known concern with these systems. This work becomes more complex with the reliability issues of these middleware components [34]. When Trident is expected to interact directly with Sigiri and Opal, the workflow author is responsible for handling the complexities for each activity, including authentication mechanisms, fault tolerance and checkpointing. In the hybrid model, the user must provide perfect configuration parameters for the next workflow engine to function properly. As we discussed in section 5, the cost of failure of a workflow at a secondary workflow system can be higher compared to the remote grid service approach. This gap further widens proportional to number of nodes in the workflow.

7. Conclusion

The results of our study show that the remote workflow engine approach generally outperforms the remote grid/cloud middleware approach. This is due in part to the fact that the workflow engines we selected are currently in use in scientific experiments and so have many optimizations built into them. For example, the dynamic service creation and lifecycle management within ODE stack through the generic factory service significantly reduces the overhead in handling application service invocations. Additionally, intelligent file movement services substantially reduces overheads in data intensive workflows. We can also safely generalize the results of our performance evaluation to different test workflows. This is because the overheads we see are largely associated with initialization and termination of the components in the setup like the local workflow engine, remote workflow engine and remote grid services and these overheads will hold for different test samples, with variations only in the workflow execution time.

Grid middleware is responsible for a significant amount of overhead in a scientific workflow stack scheduling jobs into super-computing resources. The job failures and the higher variation of overheads we experienced during our evaluation suggests that the instability and unpredictable behavior of grid middleware components have high impact on the scientific workflow systems that are using them. However efficient and optimized these workflow stacks are, the issues in middleware can make these workflow suites uncertain, if not unusable.

Similar to other workflow systems, Trident facilitates workflow runs in Windows based environments. But we think more improvements are necessary to make this toolkit more useable among the scientific research community. For example, Trident lacks the support for parallel execution constructs. Even though activities are picked up and scheduled in parallel, parallel workflows are executed sequentially.

Because scientific applications are written using a wide variety of programming and scripting languages, they are often wrapped and exposed using interoperable methods in order to be invoked by workflows engines. We used two generic service factories, GFac and Opal, to wrap applications. GFac exposes an application deployed in a supercomputing resource. But with the increased usage of cloud computing platforms for scientific workflow executions we intend to expand our evaluation to include cloud computing platforms using other application wrapping methods.

8. Acknowledgements

This project was funded in part by the National Science Foundation grants NSF CSR-0720580 and NSF EIA-0202048, and a gift from Microsoft. Our thanks to Felix Terkhorn for thoughtful discussions.

References

[1] E. Deelman, Y. Gil, Managing large-scale scientific workflows in distributed environments: Experiences and challenges, in: 2nd IEEE Int'l Conf. on e-Science and Grid Computing, IEEE Computer Society, 2006.

[2] T. Oinn, M. Greenwood, et al., Taverna: lessons in creating a workflow environment for the life sciences, Concurrency and Computation: Practice and Experience 18 (10) (2006) 1067–1100.

[3] C. Catlett, The philosophy of TeraGrid: building an open, extensible, distributed TeraScale facility, in: ACM Int'l Symp. on Cluster Computing and the Grid, IEEE Computer Society, 2002.

[4] C. Goble, D. DeRoure, myexperiment: social networking for workflow-using e-scientists, in: 2nd workshop on workflows in support of large-scale science, ACM, New York, NY, USA, 2007, pp. 1–2.

[5] Z. Zhao, S. Booms, A. Belloum, C. Laat, B. Hertzberger, Vle-wfbus: A scientific workflow bus for multi e-science domains, 2nd IEEE Int'l Conf. on eScience and Grid Computing eScience06 (2006) 11–11.

[6] E. Elmroth, F. Hernández, J. Tordsson, Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment, Future Generation Computer Systems 26 (2) (2010) 245–256.

[7] R. Barga, J. Jackson, N. Araujo, D. Guo, N. Gautam, Y. Simmhan, Trident scientific workflow workbench, in: IEEE Int'l Conf. on e-Science, IEEE Computer Society, 2008, pp. 317–318.

[8] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, S. Mock, Kepler: An extensible system for design and execution of scientific workflows, in: 16th Int'l Conf. on Scientific and Statistical Database Management, IEEE Computer Society, 2004, pp. 423–424.

[9] Apache ode (orchestration director engine), http://ode.apache.org/.

[10] J. Yu, R. Buyya, A taxonomy of scientific workflow systems for grid computing, ACM Sigmod Record 34 (3) (2005) 44–49.

[11] G. Kandaswamy, D. Gannon, A Mechanism for Creating Scientific Application Services on Demand from Workflows, in: Int'l Conference on Parallel Processing Workshops, 2006, pp. 25–32.

[12] S. Krishnan, L. Clementi, J. Ren, P. Papadopoulos, W. Li, Design and evaluation of opal2: A toolkit for scientific software as a service, in: 2009 Congress on Services - I, IEEE Computer Society, Washington, DC, USA, 2009, pp. 709–716.

[13] Y. Han, A. Sheth, C. Bussler, A taxonomy of adaptive workflow management, in: Workshop of ACM Conference on Computer Supported Cooperative Work, 1998.

[14] I. Taylor, M. Shields, I. Wang, A. Harrison, The triana workflow environment: Architecture and applications, Workflows for e-Science (2007) 320–339.

[15] C. Team, Dagman (directed acyclic graph manager)Http://www.cs.wisc.edu/condor/dagman.

[16] Y. Gil, V. Ratnakar, E. Deelman, G. Mehta, J. Kim, Wings for pegasus: Creating large-scale scientific applications using semantic representations of computational workflows, in: Proceedings of the National Conference on Artificial Intelligence, Vol. 22, AAAI Press, Menlo Park, CA, 2007, p. 1767.

[17] E. Deelman, G. Singh, M. Su, et al., Pegasus: A framework for mapping complex scientific workflows onto distributed systems, Scientific Programming 13 (3) (2005) 219–237.

[18] M. Litzkow, M. Livny, M. Mutka, Condor - a hunter of idle workstations, in: 8th Int'l Conf. of Distributed Computing Systems, 1988, pp. 104–111.

[19] Z. Jinde, Study on interoperability of workflow management systems, Journal of University of Electronic Science and Technology of China 2.

[20] J. Hayes, E. Peyrovian, S. Sarin, M. Schmidt, K. Swenson, R. Weber, Workflow interoperability standards for the internet, Internet Computing 4 (3) (2002) 37–45.

[21] M. Zur Muehlen, A framework for xml-based workflow interoperability–the AFRICA project, in: Americas Conference on Information Systems, 2000.

[22] P. Kacsuk, G. Sipos, Multi-grid, multi-user workflows in the P-GRADE grid portal, Journal of Grid Computing 3 (3) (2005) 221–238.

[23] L. Wang, M. Kunze, J. Tao, Performance evaluation of virtual machine-based grid workflow system, Concurrency and Computation: Practice and Experience 20 (2008) 1759–1771.

[24] C. Stratan, A. Iosup, D. H. J. Epema, A performance study of grid workflow engines, in: 9th IEEE/ACM Int'l Conf. on Grid Computing, GRID '08, IEEE Computer Society, 2008, pp. 25–32.

[25] M. Gillmann, R. Mindermann, G. Weikum, Benchmarking and configuration of workflow management systems, in: 7th Int'l Conf. Cooperative Information Systems, 2000, pp. 186–197.

[26] H.-L. Truong, S. Dustdar, T. Fahringer, Performance metrics and ontologies for grid workflows, Future Gener. Comput. Syst. 23 (6) (2007) 760–772.

[27] I. Foster, Globus Toolkit Version 4: Software for Service-Oriented Systems, IFIP International Conference.

[28] E. Chinthaka, B. Plale, Sigiri: Uniform research abstraction for grids and clouds, to appear Concurrency and Computation: Practice and Experience.

[29] S. Shirasuna, A dynamic scientific workflow system for the web services architecture, Ph.D. thesis, Indiana University (2007).

[30] A. Goderis, C. Brooks, I. Altintas, E. Lee, C. Goble, Heterogeneous composition of models of computation, Future Generation Computer Systems 25 (5) (2009) 552–560.

[31] The open grid computing environments portal and gateway toolkit, http://www.collab-ogce.org.

[32] B. Plale, G. Fox, S. Kowalczyk, K. Chandrasekar, escience workflows 9 years out: Converging on a vision, Tech. rep., Pervasive Technology Institute, Indiana University, Bloomington, Indiana (2011).

[33] K. Droegemeier, D. Gannon, D. Reed, B. Plale, et al., Service-oriented environments for dynamically interacting with mesoscale weather, Computing in Science and Engineering 7 (6) (2005) 12–29.

[34] S. Marru, S. Perera, M. Feller, S. Martin, Reliable and scalable job submission: LEAD science gateways testing and experiences with WS GRAM on Teragrid resources, in: TeraGrid Conference, 2008.